SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

(4)

AD-A196 422

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 1042	2. GOVT ACCESSION NO	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Computational Structure of the N-body Problem		5. TYPE OF REPORT & PERIOD COVERED AI Memo
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Jacob Katzenelson		8. CONTRACT OR GRANT NUMBER(s) N00014-86-K-0180
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE April 1988
		13. NUMBER OF PAGES 54
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

[illegible]

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

N-body problem, particle simulation, tree algorithms for particle simulation, parallel computing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse side →

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 cont.

This work considers tree algorithms for the N-body problem where the number of particles is on the order of a million The main concern of this work is the organization and performance of these computations on parallel computers.

The work introduces a formulation of the N-body problem as a set of recursive equations based on a few elementary functions. It is shown that both the algorithm of Barnes-Hut and that of Greengard-Rokhlin satisfy these equations using different elementary functions. The recursive formulation leads directly to a computational structure in the form of a pyramid-like graph, where each vertex is a process, and each arc a communication link.

The pyramid is mapped to three different processor configurations: (1) A pyramid of processors corresponding to the processes pyramid graph; (2) An hypercube of processors, e.g., a connection-machine like architecture. (3) A rather small array, e.g., $2 \times 2 \times 2$, of processors faster then the ones consider in (1) and (2) above.

The main conclusion is that simulations of this size can be performed on any of the three architectures in reasonable time. 20 seconds per time step is the estimate for a million equally distributed particles using the Greengard-Rokhlin algorithm on the CM-2 connection machine. The smaller array of processors is quite competitive in performance.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo 1042 *April 1988*

Computational Structure of the N-body Problem

Jacob Katzenelson[1]

Abstract

This work considers tree algorithms for the N-body problem where the number of particles is on the order of a million. The main concern of this work is the organization and performance of these computations on parallel computers.

The work introduces a formulation of the N-body problem as a set of recursive equations based on a few elementary functions. It is shown that both the algorithm of Barnes-Hut and that of Greengard-Rokhlin satisfy these equations using different elementary functions. The recursive formulation leads directly to a computational structure in the form of a pyramid-like graph, where each vertex is a process, and each arc a communication link.

The pyramid is mapped to three different processor configurations: (1) A pyramid of processors corresponding to the processes pyramid graph; (2) An hypercube of processors, e.g., a connection-machine like architecture. (3) A rather small array, e.g., $2 \times 2 \times 2$, of processors faster then the ones consider in (1) and (2) above.

The main conclusion is that simulations of this size can be performed on any of the three architectures in reasonable time. 24 seconds per time step is the estimate for a million equally distributed particles using the Greengard-Rokhlin algorithm on the CM-2 connection machine. The smaller array of processors is quite competitive in performance.

Keywords: N-body problem, particle simulation, tree algorithms for particle simulation, parallel computing.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects agency of the Department of Defense under Office of Naval Research contract N00014-86-K-0180.

[1]On sabbatical leave from the Department of Electrical Engineering, Technion — Israel Institute of Technology.

Computational Structure of the N-body Problem

Jacob Katzenelson[*]
Artificial Intelligence Laboratory
and
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

June 6, 1988

Contents

[*]On sabbatical leave from the Department of Electrical Engineering, Technion — Israel Institute of Technology.

List of Figures

1 Introduction

The study of physical systems by particle simulation is called the "many-body" or the "N-body" problem. Such studies are conducted in celestial mechanics, plasma physics, fluid mechanics as well as in semiconductor device simulation [1]. The simulation finds the trajectories of each particle over some time interval, given the initial position, the initial velocity, the external force and the nature of the forces that the particles exert on each other.

The number of particles used for such studies is quite large. It is estimated that to get insight into three-dimensional turbulent flow about 1 million particles are needed [2]. Thus, such simulations require intensive and prolonged computations and the question of efficiency is of great interest.

This work considers a family of algorithms for calculating the force or the potential Φ which can be called "tree algorithms". These algorithms reduce the asymptotic computational complexity from $O(N^2)$ in the naive approach, where each particle interacts with each other particle, to $O(N \log N)$ in [5] [6], and down to $O(N)$ in [7] (for particles in two dimensions) and in [9] and [11] (for particles in three dimensions). The main goal of this work is the organization and performance of such algorithms on parallel computers. In particular, it considers how such computers should be organized in order to handle tree algorithms efficiently when the number of particles is very large, and how time is divided between computation time and time for communication between processors.

The work introduces a formulation of the many-body problem as a set of recursive equations based on a few elementary functions. It is shown that the algorithms of both [6] and [7] satisfy these equations using different elementary functions. The recursive formulation leads directly to a computational structure in the form of a pyramid-like graph, where each vertex is a process, and to a SIMD computational algorithm.

The pyramid is mapped to three different processor configurations:

- A pyramid of processors corresponding to the processes pyramid graph, where each vertex is assigned a process and each arc — a communication "wire".
- An hypercube of processors ,i.e., a connection-machine like architecture [12].

- A smaller array of larger processors , e.g., $2 \times 2 \times 2$ processors.

Computation and communication time are computed for the first two cases and estimates are derived for the third case.

The main qualitative conclusion is that large simulations (about 1M particle) can be performed on any of the above architectures in reasonable time. This is particularly interesting with respect to the connection machine - an existing, ready to be used system - but also noteworthy with respect to the array of processors which is quite competitive in performance.

2 A Recursive Formulation of the N-body Problem

We consider N point particles distributed in two or three dimensional space (2D or 3D). We are interested in the forces that these particles exert on each other. We use celestial mechanics terminology (point masses and Newtonian gravitational fields) although the method is equally applicable to electrostatics (electrical charges and electrostatic fields) and fluid mechanics (vortex particle and vortex field) [1].

The force that a particle at a point x, $x \in R^3$, exerts on a particle y, $y \in R^3$, is given by

$$f_{xy} = G \frac{M_x M_y}{\| x - y \|^2} 1_r \tag{1}$$

where 1_r is a unit vector in the $x - y$ direction. The forces are additive, i.e., the total force on any particle p, F_p, is the sum of the forces due to all other particles

$$F_p = \sum_{q \in particles, q \neq p} f_{pq} \tag{2}$$

An additional valuable property of these forces is the symmetry, $f_{pq} = -f_{qp}$. It is also convenient to express the forces in terms of a potential function, Φ, such that the force on a unit mass at x is

$$F_x = -grad\ \Phi \tag{3}$$

The tree algorithm proceeds by dividing the space into "computational boxes". The method is illustrated by its 2D version which is simpler to explain; the generalization to 3D is straightforward.

We choose a (2D) rectangle such that all the particles are included in it. This is called the *top* rectangle. It is partitioned into four equal rectangles, called the *children* of the *top* rectangle. *Top* is called the *parent* of the *children* rectangles. Each child is partitioned to four equal rectangles. This process continues to an arbitrary level, say h. For compatibility with prior work [11], the level of top is named the 0 - level, so the bottom level becomes the h-th level.

The set of *neighbors* of a computational box b comprises all boxes of the same size whose boundary has at least one point in common with the boundary of b. Thus, in 2D a box has at most eight neighbors; in 3D a box has 26 neighbors.

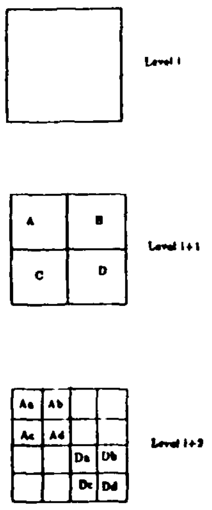

Figure 1: The computational box, its parent, children and neighbors; Aa,Ab, Ac,Ad are children of A; A is a parent of Aa,etc.;The neighbors of Ad are: Aa,Ab, Ba, Bb,Da,Cc,Ca, and Ac.

It is convenient to partition the force, or the potential, into the *far field* and the *near field* so that $\Phi_p = \Phi_{p,nearfield} + \Phi_{p,farfield}$. The reasons for this partition are twofold: (1) Approximation formulas valid for the far field potential are not valid for the near field potential; (2) Considerations related to the finite computer word size and to global accuracy necessitate modification of the near field at small distances (smoothing [3] [4]). In partitioning to far and near fields we follow [7,9] by taking the field at computational box due to particles in itself and in its neighbor boxes to be the near-field potential; the field in the box due to all boxes which are not neighbors is the far-field potential.

Let Φ_n be the far-field potential function due to particles in the box n. For any x, $x \notin n$ and $x \notin neighbors(n)$, $\Phi_n(x)$ is the contribution of particles in n to the potential at x.

Let Ψ_n be the far-field potential in n; i.e., for any x, $x \in n$, $\Psi_n(x)$ is the contribution of all particles q, $q \notin n$, and $q \notin neighbors(n)$ to the potential at the point x.

A recursive relation among Ψ_n and $\Psi_{parent(n)}$ is the heart of the tree methods:

$$\Psi_n = \Psi_{parent(n)} + \sum_{i \in I_n} \Phi_i \tag{4}$$

Where I_n, the set on which the summation of Φ_i is done, is called the *interaction set* of n and is defined by

$$I_n = \{j | j \in children(neighbors(parent(n))), j \notin neighbors(n)\} \tag{5}$$

A solution of 4 is defined by its boundary conditions

$$\Psi_{top} = 0, \tag{6}$$

and by

$$\Phi_n = \sum_{i \in children(n)} \Phi_i \tag{7}$$

where n is not a bottom level rectangular; for a bottom level n

$$\Phi_n = \Phi_n^0, \tag{8}$$

where Φ_n^0 is the far-field potential function given in terms of the particles in n.

3 The Barnes-Hut and the Greengard-Rokhlin Methods

The Barnes-Hut (BH in the sequel) [6] and the Greengard-Rokhlin (GR) [7,9] methods both solve the above recursive equations. Their differences are in how the potential functions are *represented* and *approximated*.

For presenting the material in this section it is useful to distinguish between a function, say, f, its approximation, $\hat{f}$ and the representation of this function or its approximation, $rep(f)$ or $rep(\hat{f})$. Usually, we want to represent the function itself, however, for various reasons we carry only enough information to define the approximation completely. Thus, by design, $rep(\hat{f})$ represents f.

In the BH method the function Φ_n^h is represented by two quantities $(m,\vec{c})$ where m is the sum of the masses of the particles in the box n, and $\vec{c}$ is the center of mass vector. Thus, Φ_n^h at a point y, $y \notin neighbors(i)$ and $y \notin i$, is approximated by $\hat{\Phi}_n^h$ which is the potential of a mass m at $\vec{c}$ evaluated at the point y. I.e.

$$\hat{\Phi}_n^h(y) = G\frac{m}{\| y - c \|} \tag{9}$$

Similarly, $\hat{\Phi}_n$, for level l, $l \neq h$, approximates Φ_n. It is calculated by summing the masses of the children and finding a center of mass of the masses of the children. The representation of $\hat{\Phi}_n$ is

$$rep(\hat{\Phi}_n) = (\sum_{i \in children(n)} m_i, \frac{\sum_{i \in children(n)} m_i \vec{c_i}}{\sum_{i \in children(n)} m_i}) \tag{10}$$

The approximation to Ψ_n, $\hat{\Psi}_n$, is derived from 4; $\hat{\Psi}_n$ is represented as a *list* of pairs; the pairs $(m_i, \vec{c_i})$ of all the members of interactive set are appended to form a list which, in turn, is appended with a similar list representation $\hat{\Psi}_{parent(n)}$.

The actual potential, or the force at a point p, is calculated by evaluating the appropriate $\hat{\Psi}_n$, n at level h, by evaluating each member of the list at p, and by adding the near-field effect to the result.

The 3D version of the GR method is described in [9] and in [11]. The much simpler 2D version suffices for showing how the algorithm fits the above computational structure.

In the sequel we identify R^2 with the complex plane C which simplifies the notations. Let n be a box on some level. For all z, $z \notin n$ and $z \notin neighbors(n)$, the GR method considers Φ_n as its multipole expansion around the center of n

$$\Phi_n(z) = a_0 \log(z) + \sum_{k=1}^{\infty} \frac{a_k}{z^k}. \tag{11}$$

For all z, $z \in n$, the GR method considers Ψ_n as its local (Taylor) expansion around the center of n,

$$\Psi_n(z) = \sum_{l=0}^{\infty} b_l z^l. \tag{12}$$

In both cases the functions are approximated and represented by their first $p+1$ coefficients, where p is chosen according to the accuracy required.

To derive $\hat{\Phi}_n$ from the $\hat{\Phi}$'s of its children, equation 7, the child representation is "translated", i.e., each child Φ_j, $j \in$ *children*(n) is expanded to a multipole expansion around the center of n. Once this is done, the Φ's of all children are expanded around the same origin and Φ_n is found by summing the coefficients of the translated Φ_j.

To derive Ψ_n, (equation 3) the Φ_i, $i \in$ *interactive set of* n, are converted to a *local* expansion around the center of the box n. $\Psi_{parent(n)}$ is in this local form and, therefore, the remaining operation is to translate (shift) the local expansion from the center of *parent*(n) to the center of n. Now, all these functions are represented in n by their local expansions around the same point, and Ψ_n is obtained by summing up the coefficients of all these local expansions.

Details of the GR algorithm follow:

Let n be a box of level h (bottom level). Let the center of the box be the coordinate system; let the box have m point masses $\{q_i,\ i = 1, ..., m\}$ located at points $\{z_i,\ i = 1, ..., m\}$ in the box; let r be the diagonal of the box , thus, $|z_i| \le r$. Then, for any $z \in C$ with $|z| > r$,

$$\Phi_n^h(z) = a_0 \log(z) + \sum_{k=1}^{\infty} \frac{a_k}{z^k}, \tag{13}$$

where

$$a_0 = \sum_{i=1}^{m} q_i \qquad \text{and} \qquad a_k = \sum_{i=1}^{m} \frac{-q_i\,(z_i)^k}{k}.$$

Φ_n is approximated by the first p terms of the infinite series. It can be shown that for any $p \ge 1$,

$$\left|\Phi_n(z) - a_0 \log(z) - \sum_{k=1}^{p} \frac{a_k}{z^k}\right| \le \frac{A}{1 - \left|\frac{r}{z}\right|} \left|\frac{r}{z}\right|^{p+1},$$

where

$$A = \sum_{i=1}^{m} |q^i|.$$

Clearly, $\hat{\Phi}_j$ is represented by $\{ a_k, k = 0, ..., p \}$ and the coordinates of the center of the n-th box.

To find Φ_i, i not of level h, one uses equation 4. To sum the Φ_j, $j \in$ *children*(i), the expansion representing Φ_i is "translated"; i.e., each Φ_j is expanded to a multipole expansion around the center of i. Once this is done, the Φ's of all children are expanded around the same origin and Φ_i is found by summing the coefficients of the translated Φ_j.

Let the coefficients $a_i, i = 0, ..., \infty$, specify the multipole expansion of Φ_n around the point z_0. The translation of this expansion to the origin is given by

$$\Phi_n(z) = a_0 log(z) + \sum_{l=1}^{\infty} \frac{b_l}{z^l}, \tag{14}$$

where

$$b_l = -\frac{a_0 z_0^l}{l} + \sum_{k=1}^{l} a_k z_0^{l-k} \binom{l-1}{k-1}. \tag{15}$$

with $\binom{l}{k}$ the binomial coefficients.

$\hat{\Psi}_n$ is formed by retaining the first $p + 1$ terms of the above series. It is represented by $\{ b_k, k = 0, ..., p \}$ and the coordinates of the center of the n-th box.

Ψ_n (equation 3) is computed by converting the Φ_i, $i \in$ *interactive set of* n, to a *local* expansion around the center of the box n. This conversion is given by the following:

Let $\{ a_k, k = 0, ... \}$ be the representation of some Φ_j, j is not a neighbor of n. Let n's center be the origin, and let the center of j be z_0. The contribution of Φ_j to Ψ_n at a point z in n is

$$\sum_{l=0}^{\infty} b_l z^l, \tag{16}$$

where

$$b_0 = a_0 log(-z_0) + \sum_{k=1}^{\infty} \frac{a_j}{-z^k}, \tag{17}$$

and

$$b_l = \frac{a_0}{l.z_0^l} + \frac{1}{z_0^l}\sum_{k=1}^{\infty}\frac{a_k}{-z^k}\binom{l+k-1}{k-1}. \tag{18}$$

The shifting of $\Psi_{parent(n)}$ to the center of n is given by the following equation: For any z_0, z and a_k, $k = 0, 1, ..., p$,

$$\sum_{k=0}^{p} a_k(z-z_0)^k = \sum_{i=0}^{n}(\sum_{k=l}^{n} a_k \binom{l}{k}(-z_0)^{k-i})z^l. \tag{19}$$

Several functions can be defined to summarize the GR algorithm:

Let '+' be an operator that maps representations of the same type which are expansions around the same point z into a representation around z each of its coefficient is the sum of the appropriate coefficients of the '+"s arguments.

$Translate_z$ maps a representation of a multipole expansion to a multipole expansion around the center of box z (equation 14).

$Shift_z$ maps a representation of a local expansion to a local expansion around the center of box z (equation 19).

$Convert_z$ maps a representation of a multipole expansion to a local expansion around the center of box z (equation 16).

Under these conventions the computation process by which the GR algorithm computes equations 4 and 7 can be written in the following way:

$$rep(\hat{\Phi}_n) = \sum_{i \in children(n)} Translate_n(rep(\hat{\Phi}_i)) \tag{20}$$

$$rep(\hat{\Psi}_n) = Shift_n(rep(\hat{\Psi}_{parent(n)})) + \sum_{\substack{i \in \{j | j \in children(neighbors(parent(n))), \\ j \notin neighbors(n)\}}} Convert_n(rep(\hat{\Phi}_i)) \tag{21}$$

The operations of the BH algorithm can be described with functions bearing the same names with one minor difference: $Translate_z$ has all the representations of the children of z as arguments and the summation is omitted. I.e., equation (20) becomes

$$rep(\hat{\Phi}_n) = Translate_n(rep(\hat{\Phi}_1), rep(\hat{\Phi}_2), ..., rep(\hat{\Phi}_i), ...) \quad all\ i,\ i \in children(n) \tag{22}$$

4 The Computational Structure

This section considers a network of processes, each associated with a region of space (a cell, a rectangular for 2D, a box for 3D). Each process communicates with its parents, its children and its neighbors to compute the solution to the N-body problem according to the above algorithm. If a processor is assigned to each process and all processors operate in a SIMD style, this architecture is "connection machine like" [12] in the sense that there are many equal processors operating in SIMD style which communicate via many connections to "neighbors". The difference between the connection machine and this network is that this network is not a hypercube; instead, its connections are derived from the structure of the equations. Our purpose is to find bounds on the computation time of this network architecture in order to compare it with the connection machine and other architectures.

A look at the equations shows us the following communication paths for each process (cell) C:

a. C to C's parents (equation 8),

b. C's parents to C (equation 4),

c. C to each of its children (equation 4),

d. Each child of C to C (equation 8),

e. C to C's interactive set (equation 4),

f. Each member of C's interactive set to C (equation 4),

h. C to its neighbors (h level only for the calculation of the near-field),

i. Each neighbor of C to C (h level only for the calculation of the near-field).

If we abstain from sending messages in both direction at the same time (as long as the SIMD discipline is maintained this does not occur) a and b, c and d, and h and i can be satisfied by the same "wire". Moreover, the interactive set is symmetric, i.e. for any box a, b, $a \in$ *interactive set*(b) implies $b \in$ *interactive set*(a). Therefore, e and f can also be united.

Thus, we need the following connections at each process C:

a. C to/from C's parents (8),

b. C to/from each of its children (4),

c. C to/from C's interactive set (4),

d. C to/from its neighbors (h level only for the calculation of the near-field).

If each connection is implemented, connection machine style, by one wire, we have 32 wires in 2D (1 parent, 4 children, up to 27 members in interactive set). In 3D we have 198 wires (1 parent, 8 children, up to 189 members in interactive set). This number seems high; certainly it cannot be implemented as one physical wire per connection. However, we can reduce the number of connections as follows.

The information sent from C to its interactive set members is the same as that sent to its parents. C's interactive set members can be reached by having C's parents forward the information to their (26) neighbors, who,in turn, forward it to their children. Since each message carries its source, the children can check if the sender, C, is in the child's interaction set and discard the information on a negative answer.

This argument replaces connection to interactive set members with connections to neighbors. The number of connections per cell becomes 13 for 2D and 35 (1 parent, 8 children, 26 neighbors) for 3D, which, although not low, is much better. In such an arrangement there are connections to parents, to children and to neighbors for all processors, including those at the h level. Figure 2 illustrates the connection structure in 2D, the equivalent 3D structure is somewhat difficult to draw.

Figure 3 shows that the computation structure has the form of a graph where each node describes a cell and each branch describes a wire connecting two cells. The graph has a form of a pyramid with the top node corresponding to the top cell and the bottom nodes to the h level cells. Levels, or cells of equal size, appear distinctly in the graph; they are nodes of equal height from the bottom or the top of the pyramid.

Consider an algorithm that performs the far-field computation in the following steps:

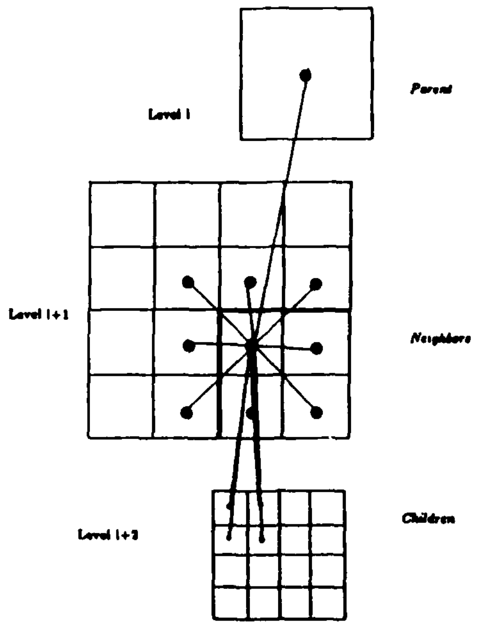

Figure 2: A 2D cell (node) and its connections

Algorithm:

1. $rep(\hat{\Phi}^0_n)$: Compute $rep(\hat{\Phi}^0_n)$, for all n, n in level h. (The exact representation and method of computation depends on the method used and will be elaborated on in the sequel.)

2. Calculate $rep(\hat{\Phi}_n)$ going up the tree: Given $rep(\hat{\Phi}^0_n)$, and using equations 8 and 20 compute $rep(\hat{\Phi}_n)$ for all n, level by level, starting in level $h-1$ and ending in level 0.

3. For all nodes of the tree calculate the effect of the interactive set members: For all direction d in a level do

   (a) Each node n sends to its interactive set member that lies in direction d from it the $rep(\hat{\Phi}_n)$ for all p, p a child of n;

   (b) Each node n receiving such an message plays the role of a neighbor and retransmits it to all its children;

   (c) Each node n (playing the role of a child) computes $Convert_z(rep(\hat{\Phi}_i))$ for all messages received from one of its interactive set members

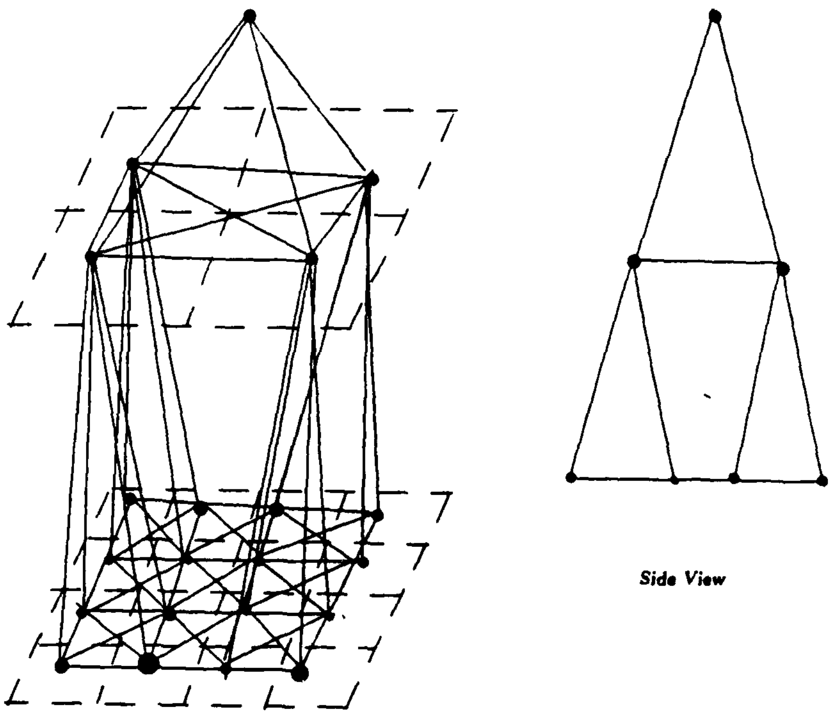

Figure 3: The computation "tree".

and discards the rest of the messages. The result is accumulated, awaiting the arrival of $rep(\hat{\Psi}_i)$ from n's parent.

4. $rep(\hat{\Psi}_n)$: The values of $rep(\hat{\Psi}_n)$ are propagated down the tree to compute $rep(\hat{\Psi}_n)$ for all children according to equation 21.

5. Evaluation: For all n in level h, $rep(\hat{\Psi}_n)$ is used to calculate the far-field potential at each mass-point in the cell.

Note that all the h level nodes can perform step 1 simultaneously. Similarly, step 2 is simultaneously performed by all nodes of the same level. Step 3 is simultaneously performed in parallel by all nodes of the entire network. This holds if we understand that a node without children does not send them messages, etc.

Some additional consideration of the algorithm shows that our tree consists of three kinds of nodes: the top node, the intermediate nodes, and the bottom level nodes. Figure 4 illustrates the functions computed and the information flow in each node.

The above algorithm differs from the algorithms described in [9] and in [11] in several aspects. First, the algorithm is really a frame in which

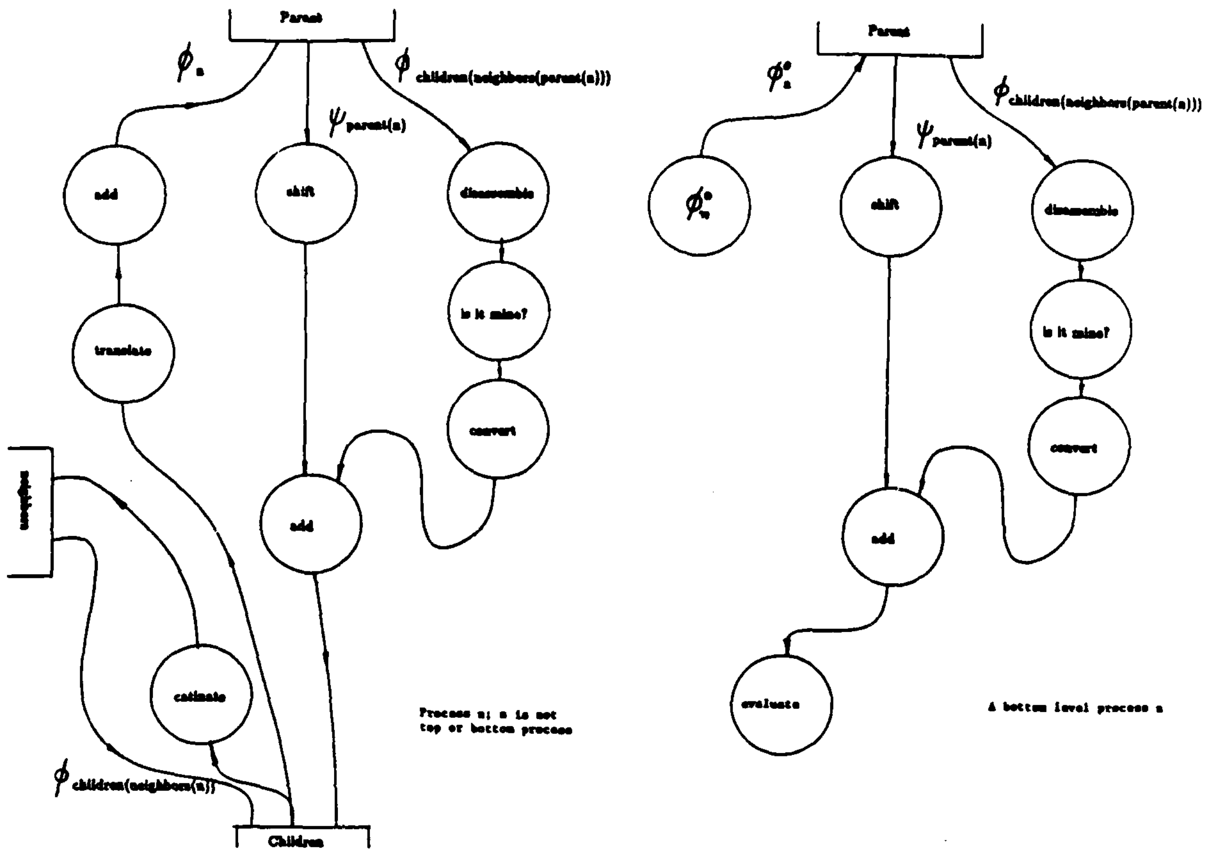

Figure 4: Computation and Information transfer in tree nodes.

different functions may be substituted for Φ_n, $Translate_z$, etc. The algorithm is described as a "tree" of processes operating in parallel rather than a single process algorithm as in [9], or the level-by-level parallel algorithm as in [11](page 50). More importantly, this algorithm handles the information transfer explicitly and so reduces both communication and computation times. The main tool for accomplishing this is the parallel structure. Information is sent in parallel so that messages do not conflict and computing is done in parallel by all processes of the tree.

The network of processes can operate also in the "data flow" mode where each process performs the computation once the data is available. The operation of the network in this mode has not been fully investigated as yet.

5 Communication Time

The time required to solve the N-body problem by the above computation structure consists of the time that each cell spends computing plus the time required to transfer the information from process to process. The later component is named *communication time.* The communication time is bounded under several assumptions.

1. Time is sampled and divided into intervals; during one time-interval a process can both send one message and receive one message.

2. Messages are of bounded length.

3. The time to send a message is one "message time" unit.

When the messages are small, one can add the assumption that processes can combine several messages into one message. It turns out that in our applications sizes are such that this assumption should be be omitted.

It is further assumed that each function representation is transferred by one message (this assumption will be modified later on). Under the above assumptions we get the following communication time bound (Notice that under the SIMD assumptions all processes are doing the same operation at the same time.)

Consider the algorithm for far-field computation in the previous section. Let us assume that in 2D there are $n \times n$ regions ($n \times n \times n$ regions for 3D) with $h = \lceil \log n \rceil$. Denote by $n_{children}$ the number of children of a processor (processor level is not h). Denote by n_n the maximum number of neighbors of a processor. Denote by n_{set} the maximum number of members in an interactive set. The following lists the *communication time* in message-time units for each step of the algorithm:

Step 2 Since each processor can send one one message and receive one message at a time, the time is $n_{children} \times h$.

Step 3 Here each processor sends $n_{children}$ messages to each neighbor which, in turn, sends one copy to each of its children. The time is $n_n(n_{children} +$

$n^2_{children}$). Note that messages can be sent and received simultaneously by all processors without interference provided that each processor sends its message in the same "direction"; i.e. all processors send to the left and receive from right, etc.

Step 4 Here each processor sends a message to each of its children. The time is $n_{children} \times h$.

Thus, the total communication time in message time units is:

$$communication\ time \leq 2h.n_{children} + n_n \times (n_{children} + n^2_{children}). \qquad (23)$$

In terms of actual time what is "message time"?

The GR algorithm specifies a function by p coefficients and by the coordinates of the center of the box around which the function is expanded (the source of the message). Since messages are function descriptions, all messages are of equal length and the concept of message time is well defined as the time required to send a function description. For example if $p = 10$ and each coefficient and the source are represented by 32 bits, the length of the message is about 352 bits. The connection machine messages, for example, are 190 bits long; therefore, on that machine, two actual messages are needed to implement our 352bit message and the "message time" becomes the time for two connection machine messages.

The case of the BH messages is somewhat more complicated. Fixed size messages, each consisting of $(center\ of\ mass, mass, source)$ travel up the tree. Therefore, the contribution of steps 2 and 3 of the algorithm is as above, totaling

$$h.n_{children} + n_n \times (n_{children} + n^2_{children}). \qquad (24)$$

However, on the way down, the number of such triplets that pass each node of the tree equals the number of interactive set members of all the node ancestors. Moreover, this function description is not a message of fixed length and the estimation of communication time has to be done differently.

The number of triplets that a node at the bottom of the tree (pyramid) receives is (less then)

$$n_{set} \times h. \qquad (25)$$

If messages are transmitted starting from the bottom of the tree, the above is the number of messages step 4 requires. It is an upper bound since some nodes have small interactive sets (e.g, *top* has zero members). Thus, the total number of messages in the BH algorithm is:

$$h \times n_{children} + n_n \times (n_{children} + n^2_{children}) + n_{set} \times h. \qquad (26)$$

With k triplets per message, the communication time is the above expression over k.

The following argument supports the above expression:

Choose an arbitrary node, say A. A sends down the tree the triplets of its interactive set. As soon as the last triplet clears A's child, the child can send its own interactive set triplet down the tree. Thus, for a tree of height h and a bound on the number of members in an interactive set, we get the above expression as a bound on the communication time.

6 Computation Time

This section estimates the computation time resulting from the arithmetic operation only. It does not include the communication time and it does not include data transfers inside processors. The estimation is made for both 2D and 3D using both GR and BH algorithms.

The time to perform floating point addition, multiplication, etc. is denoted by t_+, t_*, etc. $\hat{t}_+$, etc. denotes the corresponding time to perform the corresponding complex arithmetic. It is assumed that $\hat{t}_+ = 2t_+$, and that $\hat{t}_* = 4t_* + 2t_+$.

6.1 GR Algorithm in 2D

It is assumed that we maintain $p+1$ terms in each expansion.

6.1.1 $\hat{\Phi}_n^0$

This calculation follows Theorem 2.1.1 in [9].

It is assumed that in each cell there are m mass points q_i at points $z_i, i = 1...m$.

1. The calculation of z_i^k, $i = 1,...m$, $k = 1,...,p$, requires $m(p-1)\hat{t}_*$.
2. The calculation of a_k requires:

   For $k = 0$ the time is $(m-1)t_+$; For $k \neq 0$ the time is $(m-1)\hat{t}_+ + 2m\hat{t}_*$;
3. Summing the above and replacing the complex arithmetic entries by their equivalent floating point arithmetic:

$$(m-1)t_+ + 2(m-1)p\hat{t}_+ + m(3p-1)\hat{t}_*) = (8mp-2p-m-1)t_+ + m(12p-4)t_* \tag{27}$$

6.1.2 *Translate$_z$*

This calculation follows Lemma 2.2.1 in [9], equation 15 above. Notice that at each level there are only four different values for z_0. These differ from each other by multiplication by j (the complex rotation by $\frac{\pi}{4}$). As one goes up a level the corresponding z_0 is multiplied by 2. Thus, the values of z_0^l,

$l = 1, \ldots, p$ can be assumed to be precalculated., and 15 takes the following format:

$$b_l = -\frac{a_0(j^m 2^{h-level} z_0)^l}{l} + \sum_{k=1}^{l} a_k (j^m 2^{h-level} z_0)^{l-k} \binom{l-1}{k-1}. \tag{28}$$

where *level* denotes the cell level, and $m = 0, 1, 2, 3$.

1. The calculation of b_l requires:

   For $l = 0$ — no operations required; For $l \neq 0$ the time is $3l\hat{t}_* + l\hat{t}_+$.

2. Summing the above and replacing the complex arithmetic entries by their equivalent floating point arithmetic:

$$3\frac{p^2 + 3p}{2}\hat{t}_* + \frac{(p^2 + p)}{2}\hat{t}_+ = 4(p^2 + p)t_+ + 6(p^2 + 3p)t_*. \tag{29}$$

6.1.3 *Convert$_z$*

This calculation follows Lemma 2.2.2 in [9], equation 18 above. Notice that here , as in the above section, at each level there are only several different values for z_0; each corresponds to the position of a cell relative to each of its interactive set members. Here too, as one goes up a level the corresponding z_0 is multiplied by 2. Thus, the values of z_0^l, $l = 1, \ldots, p$ can be assumed to be precalculated; the number of sets of numbers equals the maximum number of members in an interactive set. With z_0^l assumed known constants, and taking into account the multiplications by powers of 2, we get the following operation counts:

1. The calculation of b_l requires:

   For $l = 0$ — $(2 + 2p)\hat{t}_* + p\hat{t}_+$. For $l \neq 0$ — $(2 + 2p)\hat{t}_* + p\hat{t}_+$.

2. Summing the above and replacing the complex arithmetic entries by their equivalent floating point arithmetic:

$$(2+4p+p^2)\hat{t}_* + (p+1)p\hat{t}_+ = 4(2+4p+p^2)t_* + (2.(2+4p+p^2) + (p+1)p)\hat{t}_+ \tag{30}$$

$$= (8 + 16p + 4p^2)t_* + (4 + 9p + 3p^2)\hat{t}_+ \tag{31}$$

6.1.4 *Shift$_z$*

This calculation follows Lemma 2.2.3 in [9], equation 19 above. As in the *Translate$_z$* case, at each level there are only four different values for z_0 differing from each other by multiplication by j (the complex rotation by $\frac{\pi}{4}$). As one goes up a level the corresponding z_0 is multiplied by 2. Thus, again, the values of z_0^l, $l = 1, ..., p$ can be assumed to be precalculated.

From 19 follows that

$$c_l = \sum_{k=l}^{p} a_k \binom{l}{k} (-z_0)^{k-l}. \tag{32}$$

1. The calculation of c_l requires:

$$3(p-l)\hat{t}_* + (p-l)\hat{t}_+. \tag{33}$$

2. Summing the above and replacing the complex arithmetic entries by their equivalent floating point arithmetic:

$$\frac{(p+1)p}{2}(3\hat{t}_* + \hat{t}_+) = \frac{(p+1)p}{2}(12t_* + 8t_+) \tag{34}$$

6.2 GR Algorithm in 3D

The following calculations follow the 3D expansion by F. Zhao [11]. Zhao expands the various functions to a series of the derivatives of $\frac{1}{R}$ where R is the distance from the center of a cell (whose function is being computed) to the point at which the potential is desired. To avoid evaluation of (direction) cosines, they have been expressed in terms of the cartesian coordinates (x_1, x_2, x_3). In the sequel, p is the order of the highest derivative retained. The number of terms in the expansion, denoted by n_p, is derived from p.

6.2.1 $\hat{\Phi}_n^0$

This calculation follows Theorem 3.2.1 in [11].

It is assumed that in each cell there are m mass points q_i at points (x_{1i}, x_{2i}, x_{3i}), $i = 1...m$.

$$a_{ijk} = \sum_{l=1}^{m} q_l (-1)^{i+j+k} \frac{1}{i!j!k!} x_{1l}^i x_{2l}^j x_{3l}^k. \tag{35}$$

If we retain only terms of $i + j + k \leq p$ then the number of terms is

$$n_p = \sum_{k=0}^{p} \sum_{j=0}^{p-k} \sum_{i=0}^{p-k-j} 1 = \frac{(p+1)(p+2)(p+3)}{6} \tag{36}$$

which results in $n_1 = 4$, $n_2 = 10$, $n_3 = 20$, $n_4 = 35$, etc.

1. Consider the following recursive relations between the coefficients:

$$a_{0,0,0} = 1 \tag{37}$$

$$a_{i,j,k} = a_{i-1,j,k}\frac{(-1)x_i}{i} = a_{i,j-1,k}\frac{(-1)x_j}{j} = a_{i,j,k-1}\frac{(-1)x_k}{k} \tag{38}$$

2. There are n_p recursive steps each corresponding to a different a_{ijk}.
3. Each recursive step requires $3t_*$ (i is stored as $\frac{1}{i}$).
4. The above is repeated for m particles and accumulated, giving the result $(m-1)t_+$.
5. Summing all up results in the following bound

$$n_p(3mt_* + (m-1)t_+) \tag{39}$$

6.2.2 *Translate$_z$*

This calculation follows from Lemma 3.2.1 in [11]. The coefficients of the translated expansion, c_{ijk}, are

$$c_{ijk} = \sum_{\alpha=0}^{i} \sum_{\beta=0}^{j} \sum_{\gamma=0}^{k} a_{\alpha\beta\gamma} a'_{i-\alpha,j-\beta,k-\gamma}. \tag{40}$$

Here a'_{ijk} are the coefficients in the expansion for $1/R$, measured from the old cell center with respect to the new center.

1. The a'_{ijk} are constants that can be precomputed The number of different sets of such constants equals the number of children of a cell — 8 in 3D. The constants at different levels can be derived by multiplying by appropriate powers of 2.

2. The number of terms in c_{ijk} is $(i+1)(j+1)(k+1)$.

3. Each term is a multiplication of an a',a power of 2 and an a_{ijk}; therefore each term requires $2t_* + t_+$. Thus, each c_{ijk} requires $(i+1)(j+1)(k+1)(2t_* + t_+)$.

4. Summing all up, $i + j + k \leq p$, results in

$$\sum_{k=0}^{p}\sum_{j=0}^{p-k}\sum_{i=0}^{p-k-j}(i+1)(j+1)(k+1)(2t_* + t_+) = \tag{41}$$

$$\frac{1}{720}(p^6+21p^5+175p^4+735p^3+1624p^2+1764p+720)(2t_*+t_+) \overset{\text{def}}{=} {}_p(2t_*+t_+). \tag{42}$$

The above expression (caculated by F. Zhao using MAXIMA [19])is denoted by k_p. Thus, the time is

$$k_p(2t_* + t_+). \tag{43}$$

6.2.3 *Convert*$_z$

This calculation follows from Lemma 3.2.2 in [11]. The coefficients of the converted expansion, c_{ijk},are

$$c_{ijk} = \frac{1}{i!j!k!}\sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} b_{i+\alpha,j+\beta,k+\gamma}(i+\alpha)!(j+\beta)!(k+\gamma)! \tag{44}$$

b_{ijk} are the coefficients of the local expansions for $1/R$, measured from the old cell center, with respect to the new center.

1. The b_{ijk} are constants that can be precomputed. The number of different sets of such constants in the number of interactive set members of a cell — 189 in 3D. The constants of different levels can be derived by multiplying by appropriate powers of 2.

2. Each term of c_{ijk} requires at most $(2t_* + t_+)$. (There are three items; $a_{\alpha\beta\gamma}$, a power of 2,and a constant.)

3. To calculate the contribution of all terms note that both a_{ijk} and b_{ijk} are zero if one of their coefficients is larger then p. Thus, the amount of time is

$$\sum_{k=0}^{p}\sum_{j=0}^{p-k}\sum_{i=0}^{p-k-j}\sum_{\alpha=k}^{p}\sum_{\beta=j}^{p-\alpha}\sum_{\gamma=i}^{p-\alpha-\beta}(2t_* + t_+) = \tag{45}$$

$$= \frac{1}{24}(3p^4 + 22p^3 + 57p^2 + 62p + 24)(2t_* + t_+) \overset{\text{def}}{=} c_p(2t_* + t_+) \tag{46}$$

A crude upper bound can be derived simply in the following way; If the vanishing of the b_{ijk} is not accounted for, the estimate for the number of terms of c_{ijk} is n_p; Since the number of c_{ijk}'s is n_p, the bound on time is $n_p^2(2t_* + t_+)$. The ratio between the bound of equation 46 and this last bound is about p/n_p; This illustrate the advantage in the result of equation 46.

6.2.4 *Shift$_x$*

This calculation follows from Lemma 3.2.3 in [11]. The coefficients of the converted expansion, d_{ijk},are

$$d_{ijk} = \sum_{\alpha,\beta,\gamma=0}^{\infty} c_{i+\alpha,j+\beta,k+\gamma}\binom{i+\alpha}{\alpha}\binom{j+\beta}{\beta}\binom{k+\gamma}{\gamma}\Delta x_1^{\alpha}\Delta x_2^{\beta}\Delta x_3^{\gamma}. \tag{47}$$

Here $(\Delta x_1, \Delta x_2, \Delta x_3)$ are the coordinates of the old center in terms of the new center.

1. The Δx_i are constants that can be precomputed. The number of different sets of such constants is the number of children of a cell — 8 in 3D. The constants of different levels can be derived by multiplying by appropriate powers of 2.

2. Each term of d_{ijk} requires at most $(3t_* + t_+)$ (there are four terms: c_{ijk}, a power of 2, the combinations and the Δ's).

3. Summing over all terms and all coefficients results in the following bound

$$\sum_{k=0}^{p}\sum_{j=0}^{p-k}\sum_{i=0}^{p-k-j}\sum_{\alpha=k}^{p}\sum_{\beta=j}^{p-\alpha}\sum_{\gamma=i}^{p-\alpha-\beta}(3t_* + t_+) \tag{48}$$

$$= \frac{1}{24}(3p^4 + 22p_3 + 57p_2 + 63p + 24)(3t_* + t_+) = c_p(3t_* + t_+) \tag{49}$$

6.2.5 Evaluation

The evaluation of the potential at each mass point requires the evaluation of Ψ_n, n at level h, using equation

$$\Psi_n(x) = \sum_{i,j,k=0}^{p} d_{ijk}x_1^i x_2^j x_3^k,$$

for the potential. The force can be obtained by taking the partial derivatives; Thus,

$$F_{x_1}(x) = \sum_{i,j,k=0}^{p} d_{ijk}ix_1^{i-1}x_2^j x_3^k.$$

The potential can be evaluated with two multiplications and one addition per term (keep $x_1^i x_2^j x_3^k$ for each ijk and proceed by increasing the indices one at a time). For m mass points in z, the evaluation of the potential requires

$$mn_p(t_+ + 2t_*). \tag{50}$$

For m mass points in z, the evaluation of the force requires about

$$m(n_p(t_+ + 2t_*) + 3n_pt_*). \tag{51}$$

The above was derived by assuming that first the members of the series were calculated with a power smaller by one for each x_i, and, next, for each component of the force, the series components where multiplied by the omitted components. (I.e., multiplying by ix_2x_3 for F_{x1} and ignoring the operations for forming ix_2x_3.)

6.3 BH Algorithm in 3D

The 2D BH algorithm is similar to the 3D version ; its detailed operation count is omitted.

Only the calculation of Φ_z^0 and $Translate_z$, as well as the final evaluation, have significant operation counts; $Convert_z$ and $Shift_z$ amount to concatenation of lists.

6.3.1 $\hat{\Phi}_n^0$

The following time is obtain from equation 10 (in 3D):

$$(m-1)t_+ + 3mt_* + 3(m-1)t_+ + 3t_/ = 3mt_* + 4(m-1)t_+ + 3t_/. \quad (52)$$

6.3.2 *Translate$_z$*

Translation is the above calculation with $m = 4$:

$$12t_* + 12t_+ + 3t_/. \quad (53)$$

6.3.3 Evaluation

The evaluation of the potential, or force, at each mass point requires the evaluation of each member of the list representing Ψ_n, n at level h, using equation ?? for the potential and equation 1 for the force. Ψ_n , n at level h, has at most $(h-2)*(maximum\ number\ of\ members\ in\ an\ interactive\ set)$ elements. That maximum is 189 in 3D.

The force between two points (equation 1) is evaluated according to

$$F_{x_i} = G\frac{M_x M_y}{\| x-y \|^2}\frac{x_i - x_i}{\| x-y \|}, \quad (54)$$

and requires $(3t_* + 2t_+ + 3t_-) + 15t_* + 3t_- + 3t_/$ where the 15 is an estimate of the time for evaluating a square root. With $t_- = t_+$ the result is: $18t_* + 8t_+ + 3t_/$.

Thus, for m elements and h levels, the time is:

$$189.m.(h-2)(18t_{*} + 8t_{+} + 3t_{/} + 1) \tag{55}$$

7 A Comparison

In order to gain insight into the above results this section maps the pyramid of processes to three different processor configurations:

- A pyramid of processors corresponding to the processes pyramid graph, where each vertex is assigned a process and each arc a communication "wire".
- An hypercube of processors ,i.e., a connection-machine like architecture [12].
- A smaller array of larger processors , e.g., $2 \times 2 \times 2$ processors.

The performance of the BH and the GR algorithms are compared by calculating their communication and computation times in these configurations The calculation is done for several typical values of N, the total number of particles, m number of particles in a cell, p and n_p that determine the number of coefficients that the expansion carries. To get some idea of actual time in seconds the speed of communication and floating point operations of the CM2 model connection machine is used.

7.1 The BH Algorithm - Comparative Speed

The section starts by summarizing results from previous sections.

messages: It is assumed that the BH algorithm triplet consists of 4 single precision 32bit word plus one 32bit word for specifying the origin of the cell. Thus, the 160 bit message fits into the 190 bit connection machine message; message-time unit becomes the time for one message on the CM2 connection machine which is about 0.250 millisecond.

Communication time: In 3D we have 8 children, 26 neighbors, and 189 as the maximum size of an interactive set. Thus, the communication time is $197h + 1872$ (equation 26).

Computation time: To simplify the discussion it is assumed $t_+ = t_*$; $t_/ = 5t_+$. Under these simplifying assumptions the time spent in computing is:

1. Φn^0: $(7m + 11)t_+$;

2. $Translate_z$: $39ht_+$;

3. Evaluation: $189m(h-2)(41)t_+$.

Cells and particles: Let the number of cells in the bottom level, h, be $N = n \times n \times n$; then $h = \lceil \log_2 n \rceil$. The total number of cells is:

$$N(1 + \frac{1}{8} + \frac{1}{64} + ...) \leq \frac{8}{7}N \tag{56}$$

Since the number of processors in the connection machine is $64K$, we would like to restrict the number of cells to this number.

The number of particles is mN.

Summary: The following table summarizes communication and computation times for selected values of n and m. The communication time is in message-time units and the computation time is in multiples of t_+.

n	m	*particles*	h	*communication*	$translate_z$	Φ_n^0	*evaluation*
				message units	t_+	t_+	t_+
32	1	$32K$	5	2857	195	18	23247
32	10	$320K$	5	2857	195	81	232470
32	20	$640K$	5	2857	195	151	464940
38	1	62712	6	3054	236	18	30996
38	10	627120	6	3054	236	81	309960
38	20	$1250K$	6	3054	236	151	619920

To get some insight into these numbers we recast the above table using 20Kflops per second for t_+ (CM2 model connection machine measured time [14]), and 250 microseconds per message (CM2 model connection machine message time [15]). Time in the table is expressed in milliseconds and is rounded.

n	m	*particles*	h	*communication*	$translate_z$	Φ_n^0	*evaluation*
				millisec			*millisec*
32	1	$32K$	5	714	–	–	1150
32	10	$320K$	5	714	–	–	11500
32	20	$640K$	5	714	–	–	23250
38	1	62712	6	763.5	–	–	1550
38	10	627120	6	763.5	–	–	15500
38	20	$1250K$	6	763.5	–	–	31000

The table illustrates several characteristics of this computation scheme;

1. While the communication time is substantial, it does not increase with N so long as the height of the tree is constant. Neither, in fact, do any of the quantities associated with the "tree". These quantities are a function of the size of the tree, i.e., of n and h, but not the number of particles.

2. With a low number of particles most of the time is communication time.

3. Since the number of processors in the connection machine is fixed, to reach a high number of particles m has to increase. This shifts the computation load to the evaluation and to the bottom level (h level) of processors. They do most of the work while the rest of the processors are idle.

4. The evaluation time could be reduced significantly by increasing the number of processors so that m remains 1. This cannot be done in the connection machine that has a fixed number of processors.

7.2 The GR Algorithm - Comparative Speed

The section starts by summarizing results from previous sections.

messages: It is assumed that the GR algorithm uses n_p coefficients; each coefficient is represented by a single precision 32bit word plus one 32bit word for specifying the origin of the cell. Thus, the length of a message is $32n_p$ bits. A message-time unit is the time for one such message; the exact time on any particular hardware,say, the connection machine, depends on n_p.

Communication time: In 3D we have 8 children and 189 as the maximum size of an interactive set. Thus, the communication time is $16h + 26.72$ (equation 24).

Computation time: To simplify the discussion it is assumed $t_+ = t_*$; $t_/ = 5t_+$. Under these simplifying assumptions the time spent in computing is:

1. Φn^0:

$$n_p(4m-1)t_+;$$

2. *Translate$_z$*:

$$3k_pt_+;$$

3. *Convert$_z$*:

$$3c_pt_+;$$

4. *Shift$_z$*:

$$4c_pt_+;$$

5. Evaluation: Potential:

$$3mn_pt_+;$$

Force:

$$6mn_pt_+.$$

Denote by n_n the number of neighbors and by $n_{children}$ the number of children. The total time is:

$$\tau(\Phi n^0)+h.\tau(Translate_z)+(n_n).n_{children}.\tau(Convert_z)+h.\tau(Shift_z)+\tau(Evaluation) = \tag{57}$$

$$(n_p(4m-1)+3hk_p+3n_nn_{children}c_p+4hc_p+3mn_p)t_+ \tag{58}$$

$$(n_p(7m-1)+(3k_p+4c_p)h+3n_nn_{children}c_p)t_+ \tag{59}$$

For reference in the sequel it is convenient to denote

$$P = n_p(7m-1)t_+ \tag{60}$$

which is the time spent in the leaves of the tree, and to denote

$$Q = ((3k_p+4c_p)h+3n_nn_{children}c_p)t_+ \tag{61}$$

which is the time spent in the tree nodes that are not leaves.

Cells and particles: Let the number of cells in the bottom level,h, be $N = n \times n \times n$; then $h = \lceil \log_2 n \rceil$. The total number of cells is:

$$N(1+\frac{1}{8}+\frac{1}{64}+...) \leq \frac{8}{7}N \tag{62}$$

Since the number of processors in the connection machine is $64K$ we would like to restrict the number of cells to this number.

The total number of particles is mN.

Summary: The following table summarizes communication and computation times for selected values of n, m and p. The computation time is stated in terms of t_+. The communication time is stated in terms of message units.

The time for Translate+ Convert+ Shift is denoted by Q. This time is independent of m. In the table this time is separated from the time which depends on m and which represents operations done by the bottom level processes.

n	m	*particles*	h	p	n_p	*com–munication message units*	Q t_+	$\Phi n^0 +$ *evaluation* t_+
32	1	$32K$	5	1	4	1952	4571	24
32	1	$32K$	5	2	10	1952	15732	60
32	1	$32K$	5	3	20	1952	42685	120
32	10	$320K$	5	1	4	1952	4571	276
32	10	$320K$	5	2	10	1952	15732	690
32	10	$320K$	5	3	20	1952	42685	1380
32	20	$640K$	5	1	4	1952	4571	556
32	20	$640K$	5	2	10	1952	15732	1390
32	20	$640K$	5	3	20	1952	42685	2780
38	1	62712	6	1	4	1968	4620	24
38	1	62712	6	2	10	1968	15912	60
38	1	62712	6	3	20	1968	43188	120
38	10	627120	6	1	4	1968	4620	276
38	10	627120	6	2	10	1968	15912	690
38	10	627120	6	3	20	1968	43188	1380
38	20	$1250K$	6	1	4	1968	4620	556
38	20	$1250K$	6	2	10	1968	15912	1390
38	20	$1250K$	6	3	20	1968	43188	2780

The above table is translated below to "real" numbers: the communication time assumes a 190bit message and coefficients of 32bit; the computation time assumes 20Kflops per second, typical of the CM2 model connection machine [14]. The communication time is assumed to be 250 microseconds per message [15].

n	m	*particles*	h	p	n_p	*com–munication millisec*	Q $3(2h+104)n_pp$ *millisec*	$\Phi n^0 +$ *evaluation* $(9m+7p-4)n_p$ *millisec*
32	1	32K	5	1	4	488	228	1
32	1	32K	5	2	10	$488*3$	786	3
32	1	32K	5	3	20	$488*6$	2135	6
32	10	320K	5	1	4	488	228	14
32	10	320K	5	2	10	$488*3$	786	35
32	10	320K	5	3	20	$488*6$	2135	69
32	20	640K	5	1	4	488	228	28
32	20	640K	5	2	10	$488*3$	786	70
32	20	640K	5	3	20	$488*6$	2135	139
38	1	62712	6	1	4	492	231	1
38	1	62712	6	2	10	$492*3$	796	3
38	1	62712	6	3	20	$492*6$	2159	6
38	10	627120	6	1	4	492	231	14
38	10	627120	6	2	10	$492*3$	796	35
38	10	627120	6	3	20	$492*6$	2159	69
38	20	1250K	6	1	4	492	231	28
38	20	1250K	6	2	10	$492*3$	796	70
38	20	1250K	6	3	20	$492*6$	2159	139

This table illustrates several characteristics of this computation scheme:

1. As in the BH case, the communication time does not increase with m. Neither do, in fact, any of the quantities associated with the "tree". These quantities are a function of the size of the tree, i.e., n and h, but not of the number of particles.

2. Unlike the BH case, most of the time is spent in the "body" of the tree performing *Convert* - calculating the effect of the interactive set. The evaluation stage is relatively fast.

3. The GR algorithm seems to be several times faster than the BH algorithm for a large number of particles arranged as indicated below. It requires less communication time and less computing time. For a small number of particles (say, 32K particles) the GR algorithm with $p = 1$ is

faster then the BH algorithm. An accurate comparison requires comparing performance for the same global error. Such an analysis has not as yet been done.

4. Unlike the BH algorithm, the GR algorithm utilizes all the processors more or less equally. In considering the replacement of a connection network by fewer, faster processors we can take advantage of the fact that the bottom level is significantly busier than the rest of the computing elements; using the GR algorithm all the processors have to be accounted for and replaced (more on this subject in the sequel).

7.3 Mapping the "pyramid" into a hypercube

The mapping of the network of processes into an hypercube is preparation for executing the algorithm on a connection-machine-like computer. The interest here is twofold. How fast can a hypercube machine simulate the network and how can the n-body computation be organized on the hypercube. The definitions in this section follow [17]. The actual mapping is a rather straightforward generalization of [18].

An embedding $< \phi, \rho >$ of a graph $G = (V_G, E_G)$ into a graph $H = (V_H, E_H)$ is defined by an injective mapping from V_G to V_H, together with a mapping ρ that maps $(u, v) \in E_G$ onto a path $\rho(\phi(u), \phi(v))$ in H that connects $\phi(u)$ and $\phi(v)$.

The quality of an embedding is measured by three cost functions — *expansion, dilation,and load factor*. The *expansion* of an embedding $< \phi, \rho >$ of G into H is the ratio of the size of V_H to the size of V_G. The *dilation* of an edge (u, v) under $< \phi, \rho >$ is the length of the path $\rho(\phi(u), \phi(v))$ in H. The *dilation* of an embedding $< \phi, \rho >$ is the maximum edge dilation, over all edges in G, under $< \phi, \rho >$. The *load factor* $\lambda(e)$ of an edge e in H is the number of paths that pass through e which are images of edges in G. The load factor of an embedding is defined to be the maximum load-factor over all edges in H.

Consider the 2D case first. An $n \times n$ pyramid graph is formed from meshes of size $n \times n$, $\frac{n}{2} \times \frac{n}{2}$, $\frac{n}{4} \times \frac{n}{4}$, $\frac{n}{8} \times \frac{n}{8}$, ..., 1×1 connected as implied by figure 5. Note that n is a power of 2 and that if each node is one of our cells, this pyramid does not contain the connection to the "diagonal" neighbors.

(Omission of the diagonal link does not change d in either the two or three dimensional cases. As it is shown in the sequel, d is the important quantity in our case.)

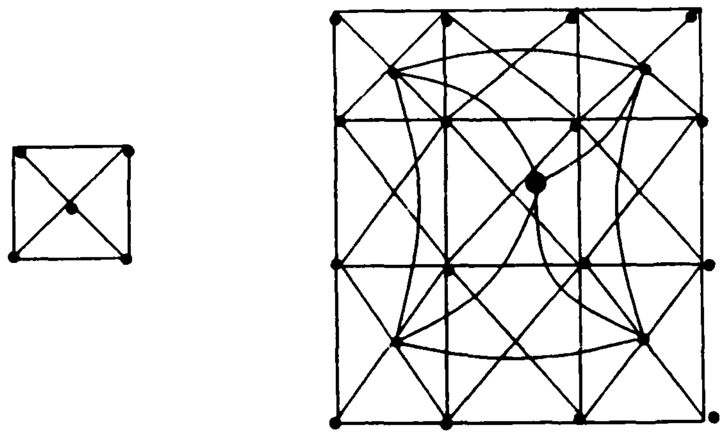
Figure 5: A 2×2 and a 4×4 pyramid

Lemma 7.1 *An $n \times n$ pyramid can be embedded in a $2n^2$ node hypercube so that dilation of the embedding is* 3 *and the load-factor is* 2.

See [18] for a proof.

Lemma 7.2 *An $n \times n \times n$ pyramid can be embedded in a $2n^3$ node hypercube so that dilation of the embedding is* 4 *and the load-factor is* 3.

The lemma can be proved similarly to the proof in [18].

Clearly, we are interested in a mapping that simulates the tree well. Since each process is mapped to a different processor, the issue is an issue of communication time rather then computation time. Maps differ from each other in the amount of communication overhead they introduce as compared to the tree. While there is no claim that the above mapping is optimal, it is

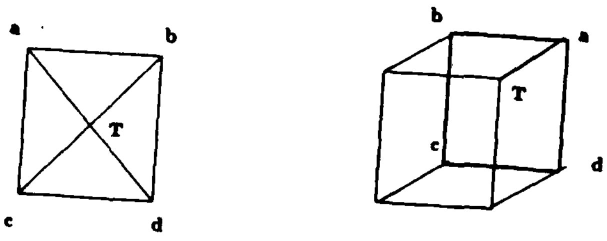

Figure 6: A simple 2D embedding.

attractive since neighboring nodes remain close together. Moreover, we can *bound the number of cycles of the hypercube machine needed to simulate* a cycle of the pyramid machine.

[17] considers the case of simulating G by H where all nodes sends messages at the same time on all adjacent wires. The resulting bounds are $max\{d, \lambda\} \leq T \leq d\lambda$ where T is the number of cycles of H required to simulate any cycle of G. These bounds are too pessimistic for our case. In our case an upper bound is simply the dilation d. This is indeed not the bound on a general computation performed on a pyramid network but a bound on our own algorithm with the above mapping. The bound results from the fact that in this mapping the only overlapping paths are those from a node to its children. Since only one child at a time sends messages to its parent, no conflict arises. Since in most part of the algorithms the information consists of several messages that can be queued inside a process, it is believed that d is actually a conservative upper bound.

The conclusion is that hypercube communication can be bounded by d multiplied by the communication performance of the pyramid. We believe that this is a rather conservative estimation of performance; with the pyramid mapped as above the average performance of the connection machine will be very close to that of the pyramid machine.

The expansion story is less attractive. A $32 \times 32 \times 32$ base (32K cells in

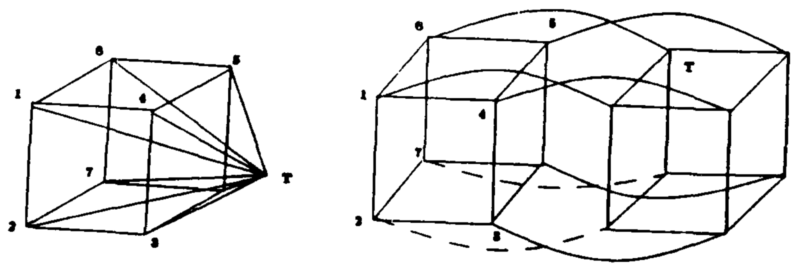
Figure 7: A simple 3D embedding.

the bottom level) requires 64K processors, $\frac{6}{14}$ of them not participating in the computation.

7.4 A Small Number of Larger Processors

This section considers the implementation of the algorithm by a small number of larger processors. Both "small number" and "larger" are with respect to the connection machine. We consider either 1 processor, or , for 3D arrays of $2 \times 2 \times 2$, $4 \times 4 \times 4$, etc. of processors. What we have in mind for a processor is a board with fast arithmetic chips, memory and a controller. Currently, a board like this can deliver about 20M floating point operations per second and has a memory of several million words of 32 bits each. The interconnection among neighboring processors is via 32-bit buses. To facilitate simple programming and symmetric information transfer between processors the buses are interconnected as illustrated by figure 8. The processors are associated with grid points. Each processor has an input bus and an output bus. The input bus of a processor is connected (via tri-state gates) to the output buses of all its neighbors. The output bus of a processor is connected (via tri-state gates) to the input buses of all its neighbors.

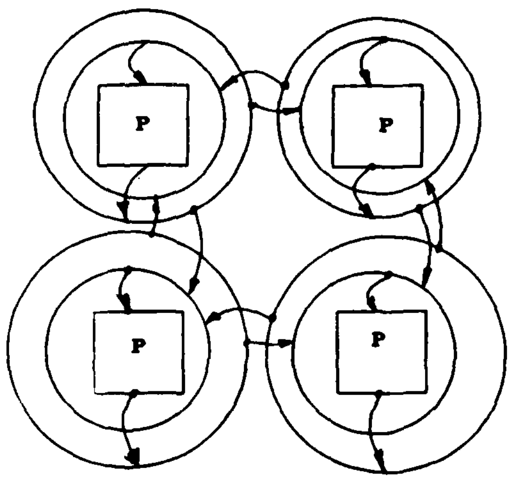

Figure 8: Interconnection among neighboring processors.

The pyramid of processes has to be partitioned and assigned to individual processors. It is assumed that this partition is symmetric; The base of the pyramid is equally divided among the processors, each getting a smaller pyramid; whatever is left is divided again, or assigned to one processor. For example, a 32×32 base pyramid of processors is divided among 4 processors; Each gets a 16×16 base pyramid; the top process remains and has to be assigned to one of the processors.

There are several differences between this computation mode and the ones discussed before. First, no messages have to be sent between two processes that resides on the same processors (*local* processes). (A process accesses the results of another process directly.) Thus, in this case, the communication time is negligible. Second, a processor evaluates its processes sequentially; parallelism exists between processors only. Thus, the evaluation order has to be such that no processor will be waiting for data from another processor.

The transfer of information between processors is different. It is assumed that each processor has a DMA channel so that data transfer to and from buses can be done simultaneously with computing. In the pyramid case, to save "wires", information was routed to interactive set members through parents. Clearly, this does not make any sense for local processes; for processes in different processors the issue is how not to send the same information more than is necessary. Therefore, when a cell has a result which is needed by a cell in another processor, the result is sent over; the other processor has a memory cell for each cell on other processor that is in one of its own cells interactive sets. The arriving information is stored there to be used by all cells that need it.

Let us consider a pyramid of processes with a base of $n \times n \times n$ (e.g., $n = 32$) and an array (of larger processors) of size $k \times k \times k$, where $k = 2, 4, 8$, etc.

The pyramid of processes is partitioned among the array's processors so that each processor gets a pyramid of base $\frac{n}{k} \times \frac{n}{k} \times \frac{n}{k}$. The top of the original pyramid is not covered; In the case of base 32 pyramid, which is used here for comparison, and arrays of base 2, 4, and 8, the top is a pyramid with base $1 \times 1 \times 1$, $2 \times 2 \times 2$, and $4 \times 4 \times 4$, respectively. It is assumed that this tip is either assigned to one of the processors or divide among them. For the cases $k = 2, 4, 8$ the contribution of the tip pyramid to the total time can be ignored and this is what is done in the sequel.

It is assumed that the systems of processors uses the same algorithm as in section 4, except that each processor services all processes in its own pyramid. The processors work in parallel and in synchronization so that no processor has to wait for the results of its fellow processors.

It is not difficult to see that each processor implements somewhat less than $\frac{8}{7}(\frac{n}{k})^3$ processes. Similarly, a processor has to implement the communication between the processes; the time for communication between processors residing on the same processor can be ignored. The communication time between processors has to be accounted for. This communication is represented by "wires" to from processes on the pyramid faces to neighboring processes which reside on neighboring processors. In 3D the number of these wires is proportional to $(\frac{n}{k})^2$ and the sequel considers their effect on the communication time.

The calculation of the time computation is based on 20M floating point operations per seconds and the communication time is based on transfer time of 80 nanoseconds per 32 bits.

Communication time: It is assumed that each processor has pseudo-cells , or p-cells, one p-cell for each cell z, such that z is a member of an interactive set of a cell in the processor's pyramid, z not in the pyramid. The number of such cells can be calculated. For a cell y, y a cell on the pyramid face, these are the $children(neighbors(parent(y)))$ which are not in the pyramid. (Note that $parent(y)$ is in the pyramid and that we are interested in $neighbors(parent(y))$ which are not in the pyramid.) If we "cover" the pyramid faces with such cells, this cover includes all the interactive set members of cells in the pyramid where the members are not in the pyramid. The number of cells in this cover is:

$$(number\ of\ children) \times (number\ of\ cells\ in\ the\ faces\ of\ a\ \frac{n}{2k} \times \frac{n}{2k} \times \frac{n}{2k}\ pyramid) \tag{63}$$

The number of cells in a face of a $n \times n \times n$ pyramid is

$$n^2(1 + \frac{1}{4} + \frac{1}{16} + ...) < \frac{4}{3}n^2 \tag{64}$$

Thus, the number of cells in the above cover becomes

$$8(\frac{4}{3}(\frac{\frac{n}{k}}{2})^2 \times 6 + 8\log(\frac{n}{2k})) \tag{65}$$

The last term are cells that cover the corners. This last term can be just ignored.

Thus, the number of p-cells is about $16(\frac{n}{k})^2$ and this is the number of messages the processor has to send or to receive (ignoring the fact that some processors do not have neighbors on all directions).

The GR algorithm defines the content of such a cell by $n_p + 3$ coefficients (the 3 represents the coordinates of the source). Thus, the communication time is

$$(n_p + 3) \times 32 \times 16(\frac{n}{k})^2(\frac{80 \times 10^{-9}}{32}) \tag{66}$$

Similarly, the BH algorithm defines the content of such a cell by $1 + 3$ coefficients (the 3 represents the coordinates of the source). Thus, the communication time is

$$4 \times 32 \times 16(\frac{n}{k})^2(\frac{80 \times 10^{-9}}{32}) \tag{67}$$

Computation time: A processor that implement a $\frac{n}{k} \times \frac{n}{k} \times \frac{n}{k}$ base pyramid has to perform all the operations that its processes do.

For the 3D GR algorithm the computation time is:

$$(\frac{n}{k})^3(\tau(\Phi n^0)+\frac{8}{7}\tau(Translate_z)+\frac{8}{7}n_{set}\tau(Convert_z)+\frac{8}{7}\tau(Shift_z)+\tau(Evaluation)) \qquad (68)$$

Using equations (39), (43), (46), (49) and (50), the expression becomes

$$(\frac{n}{k})^3(n_p(4m-1)+3\frac{8}{7}k_p+3n_{set}c_p+4\frac{8}{7}c_p+3mn_p)t_+ \qquad (69)$$

which results in

$$(\frac{n}{k})^3(n_p(7m-1)+\frac{8}{7}(3k_p+(3n_{set}+4)c_p))t_+. \qquad (70)$$

The computation time of the 3D BH algorithm almost entirely consists of the evaluation time. Thus, for a $\frac{n}{k}\times\frac{n}{k}\times\frac{n}{k}$ base pyramid the computation time is

$$(\frac{n}{k})^3(evaluation\ time\ for\ one\ processor\ of\ the\ pyramid)\frac{1}{1000}$$

GR algorithm summary: The following table summarizes the computation and communication time for the GR algorithm and for various values of n, p and k.

n	m	particles	p	n_p	k_p	c_p	k=2		k=4		k=8	
							comm-time	comp-time	comm-time	comp-time	comm-time	comp-time
							sec		sec		sec	
32	20	640K	1	4	7	7	2.3	1.05	0.57	0.132	0.15	0.016
32	20	640K	2	10	28	25	4.27	3.6	1.07	0.46	0.27	0.057
32	20	640K	3	20	84	65	7.53	9.3	1.87	1.16	0.47	0.146
38	20	1250K	1	4	7	7	3.25	1.76	0.8	0.22	0.21	0.027
38	20	1250K	2	10	28	25	6.0	6.1	1.51	0.764	0.37	0.095
38	20	1250K	3	20	84	65	10.6	15.6	2.63	1.95	0.67	0.244

The above numerical results can be compared with the the performance of the GR algorithm on the pyramid (section 7.2). The following table compares the performance for $p = 3$. Define *Ratio*

$$Ratio \stackrel{\text{def}}{=} \frac{(total\ time\ pyramid)}{(total\ time\ k\times k\times k\ array)}. \qquad (73)$$

n	m	*particles*	p	*Ratio*	*Ratio*	*Ratio*
				$k=2$	$k=4$	$k=8$
32	20	$640K$	3	0.32	1.75	8.61
38	20	$1250K$	3	0.20	1.16	5.23

BH algorithm summary: The following table summarizes the computation and communication time for the BH algorithm.

n	m	particles	k=2		k=4		k=8	
			comm-time	comp-time	comm-time	comp-time	comm-time	comp-time
			sec		sec		sec	
32	20	640K	1.3	126.4	0.33	16.0	0.08	2.0
38	20	1250K	1.84	179.2	0.46	22.4	0.11	2.4

The above numerical results can be compared with the the performance of the BH algorithm on the pyramid (section 7.1).

n	m	particles	Ratio	Ratio	Ratio
			k=2	k=4	k=8
32	20	640K	0.15	1.21	9.8
38	20	1250K	0.14	1.25	9.12

Summary: It follows from the previous section that

$$\begin{aligned} &\textit{computation time pyramid} + \textit{communication time pyramid} \leq \\ &\textit{total time hypercube} \leq \\ &\textit{computation time pyramid} + d \times (\textit{communication time pyramid}). \end{aligned} \tag{74}$$

where $d = 4$. Bounds for the ratio of the total time, hypercube, to the total time, array, can be derived from this relation. Since I believe that $d = 4$ is a rather conservative bound, I am giving the hypercube the benefit of the doubt and, for comparison sake, consider its performance equal to the pyramid.

The above numerical results indicates that the array can comfortably compete with either the pyramid or the hypercube (both with the CM2 connection machine parameters). A general comparison of the array and the hypercube has to weight two additional factors: First, the hypercube, in the form of the connection machine, is an existing commercially available equipment; the array, has to be built. Second, the smaller arrays, e.g. $2\times2\times2$, can be expected to be relatively inexpensive, probably more then a order of magnitude less expensive than the connection machine.

8 Near-field Computation

This section estimates the computation and communication time required for near-field computation. These estimates are added to the estimates of the far-field computation to yield an estimate for the total time required.

Let q be a point mass at some cell n. The near-field force on q is the force exerted on q by particles in n an in *neighbors*(n). This force is computed using

$$F_{x_i} = G\frac{M_x M_y}{\| x - y \|^2}\frac{x_i - y_i}{\| x - y \|}, \tag{75}$$

where M_x and M_y are the two mass points in 3D whose coordinates are $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, respectively, and F_{x_i} is the force on M_x in the x_i direction. This calculation is rather straight forward and the only issue is how to take advantage of the symmetry of the force ($F_{x_i} = -F_{x_i}$) which, in parallel computation, requires some elaboration.

Consider the pyramid computational structure. Only the bottom layer processes participate in the near-field computation. The following is a step by step description of this computation.

- For all cells c in the bottom layer evaluate the forces on each mass in c due to the other mass points in c.

  The evaluation of equation 75 for all force components requires $2t_+ + 6t_- + 23t_* + t_/ = 36t_+$. There are $\frac{1}{2}m(m-1)$ such computations; Under the assumptions stated in section 6 the total time is $\frac{36}{2}m(m-1)t_+$.

- Consider a cell c and its neighbors. c and each of its neighbors, say e, determine a *direction*. This direction is determined by $x_e - x_c$ where x_e is the center of the neighbor cell and x_c is the center of c. Clearly, if $x_e - x_c$ is a direction, so is $x_c - x_e$. Let D be a set of directions such that if d is a direction the either $d \in D$ or $-d \in D$ but not both.

  For all directions d in D do

  - Each bottom layer cell c sends the value and coordinates of each of its mass points to its neighbor in the d direction.

    Communication time is m message unites.

- Each cell c provides $3m$ locations for the forces on the mass points whose values and coordinates it just received. Next it calculates the forces between any two pairs of mass points, one in c and one that its values just arrived from the neighbor.

  Computation time is $36m(m-1)t_+$
- The values accumulated in the $3m$ locations are sent back to the neighbor cell from which the mass points came.

  Communication time is m message units.

Since the number of directions in D is $\frac{n_n}{2}$ the total time is

$$(\frac{36}{2}m(m-1)t_+ + \frac{n_n}{2}(36m(m-1)t_+ + 2mt_{message\ unit}) \qquad (76)$$

Since the communication time is small as compared to the computation time we (almost) get the 1/2 factor discussed in the beginning of this section. This is achieved by having one processor calculate the force and send the results back to its neighbor.

Using the CM2 connection machine parameters once again (20k floating point operations per second, 250 microseconds per message) the time for evaluating the near-field by the pyramid is given below (n_n is 26 for 3D).

m	*communication time*	*computation time*
	$n_n m$	$36(\frac{n_n}{2}+1)m(m-1)t_+$
	msec	*msec*
10	65	2,268
20	130	9,576

The array of larger processors is faring just as well in the near-field computation. As seen from the above, communication time is negligible. Since only the bottom layer processors of the pyramid participate in the calculation we get for an $n \times n \times n$ base pyramid

$$\frac{pyramid\ time}{k \times k \times k\ processor\ array\ time} = \frac{1}{(\frac{n}{k})^3 \frac{1}{speedup}} \qquad (77)$$

where

$$speedup = \frac{t_+\ for\ a\ pyramid\ processor}{t_+\ for\ a\ larger\ processor} \qquad (78)$$

The speed up is 1000 for the values used in section 7.4. Thus, the value of $\frac{\textit{pyramid time}}{k \times k \times k \textit{ processor array time}}$ is given in the following table for various values of k.

k	$n = 32$	$n = 38$
2	0.25	0.146
4	2	1.17
8	16	9.33

9 Summary and Conclusions

The work formulates the N-body problem as a set of recursive equations with boundary conditions. The numerical solution to this set of equations can be viewed as a pyramid structure of processes. There are three kinds of processes: the bottom level processes, the top level processes and all the rest. All processors are implemented by means of five functions: $\hat{\Phi}_n^0$, $Translate_z$, $Convert_z$, $Shift_z$, *Evaluate*. Both the BH and the GR algorithms have the same structure. They differ from each other in the above functions. This formulation of the N-body problem simplifies the understanding of the algorithms, their analysis and programming.

The pyramid structure is mapped into three different architectures that can be viewed as possible hardware implementations. For each mapping the amount of time for arithmetic computations and the time for communication between processors is estimated. Out of these estimates several qualitative conclusions are derived:

Parallel execution reduces *both* the BH and the GR algorithms to $O(\log N)$, where N is the number of particles; this bound covers both computation and communication time. This is somewhat of a surprise, because in the serial computation the BH algorithm is considered a $O(N \log N)$ algorithm while the GR one is $O(N)$.

The algorithms exhibit both common and differing patterns of behavior. In both cases, given the the height of the tree h, the time spent in the tree is independent of m, the number of particles in a cell. The evaluation stage, however, is hard on BH; for large m most of the time is spent in it, and computation time far exceeds communication time. For large m the GR algorithm performs exceedingly well; the evaluation of the potential function for each particle is straightforward. Clearly, using a $38 \times 38 \times 38$ pyramid, the GR algorithm can handle 10M equally distributed particles almost as fast as it handles 1M. At these numbers, the increase of m from 20 to 200 increases the total time by about 25 percent only because the algorithm is bounded by the communication time rather than by the evaluation time at the bottom cell.

Since the BH algorithm is much simpler to program it is worthwhile noting that its performance for 62K particles, one at a cell, is about equal to that of the GR algorithm when $p = 2$, using the same number of particles. A complete comparison of BH and GR requires comparing the algorithms for the same global error. This work has not been completed as yet.

A hypercube machine can simulate the pyramid structure without conflict of messages and with a slowdown equal to or less than $d = 4$.

Using the GR algorithm with $p = 3$ and a connection machine of the present size, it is possible to calculate a solution to the 1M particle (static) many body problem in about 24 seconds. This number is based on multiplying $d = 4$ by the communication time computed for the tree and adding the computation times for the far- field and for the near-field. It is considered a rather conservative estimate. This number is also the estimate for one time step for the dynamic many body problem.

For running the N-body algorithm the connection machine has a clear advantage: it is an existing product that can be readily used. The analysis, however, points out some possible disadvantages; while 64K processors seem quite a large number, the 3D N-body problem reaches the limit rather fast. One has to increase m , assigning more particles per cell. Clearly, at least for the BH case, we would rather *increase the number of processors*; a feature the hardware does not support.

Rough performance estimates for an array of faster processors indicate that with a modest size array it is possible to get performance competitive to that of the connection machine.

The above calculations considered particles that are equally distributed in space. Often, physical system exhibit sparsity, certain regions of space have many particles while others are empty or nearly so. Taking advantage of sparsity awaits farther work.

The computation and the communication times computed should be considered initial estimates as data moving operations were not taken into account. Moreover, these estimates await experimental verification, preferably by programs simulating the above processes. This work is currently underway.

Acknowledgment

Thanks are due to Gerald J. Sussman for introducing me to astronomy, Alexandre J. Chorin for bringing the work of Greengard and Rokhlin to my attention and for commenting on the manuscript, Harold Abelson and

Feng Zhao for detailed comments and discussions, Joshua Barnes, Andrew A. Berlin, J. Makino, and Piet Hut for discussions, and Charles E. Leiserson for pointers to the graph work,

This report describes research done in the Artificial Intelligence Laboratory of the Massachussetts Institute of Technology. Support for the Laboratory's artificial intelligence reseach is provided in part by the Advanced Research Project Agency of the Department of Defence under Office of Naval Research contract N00014-86-K-0180.

References

[1] R. W. Hockney and J.W.Eastwood, *Computer Simulation using Particles.* McGraw Hill, New York, 1981.

[2] A. J. Chorin, private communication.

[3] A. J. Chorin, *Numerical Studies of Slightly Viscous Flow*, J. Fluid Mech., Vol 57,785-796, 1973.

[4] C. Anderson and Claude Greengard, *On Vortex methods*, SIAM J. Nuer. Anal., Vol. 22, NO3, June 1985, pp 491-440.

[5] A. Appel, An efficient program for many-body simulation. *SIAM. J. Sci. Stat. Comput.*, 6(1), Jan. 1985.

[6] J. Barnes and Piet Hut, *A Hierarchical $O(N \log N)$ Force Calculation Algorithm.* Technical Report, The Institute for Advanced Study, Princeton, NJ 08540, 1986.

[7] L. Greengard and V. Rokhlin, *A Fast Algorithm for Particle Simulations.* Research Report YALEU/DCS/RR-495, Yale University, April 1986.

[8] J. Carrier, L. Greengard, and V. Rokhlin, *A Fast Adaptive Multipole Algorithm for Particle Simulations.* Research Report YALEU/DCS/RR-496, Yale University, January 1987.

[9] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems.* PhD thesis, Yale University, April 1987.

[10] L. Greengard and V. Rokhlin, *The Rapid Evaluation of Potential Fields in Three Dimensions.* Research Report YALEU/DCS/RR-518, Yale University, January 1987.

[11] F. Zhao, *An $O(N)$ algorithm for three-dimensional N-body simulations.* Master's thesis , MIT, Dept. of Electrical Engineering and Computer Science, Oct. 1987.

[12] D. Hillis, *The Connection Machine.* MIT Press, 1985.

[13] D. Hillis and G. Steele Jr., Data parallel algorithms. *Comm. of the ACM*, 29:1170–1183, 1986.

[14] J. Makino, Unpublished report, School Of Advanced Study, Princeton, NJ, 1987.

[15] Brewster Kahle, Private communication.

[16] Bit Bipolar Integrated Technology, Inc. Designing a Micro-Sequenced CPU with the *B3110/B3120*, AN-1, August 1987.

[17] S. Bhatt, F. Chung, T Leighton, and A. Rosenberg, *Optimal Simulation of Tree Machines,* 27th IEEE Conf on Found. of CS. Oct 1986, pp 274-282.

[18] T. Leighton, 18.435 course handouts, MIT, November 1987.

[19] *Macsyma Reference Manual, version 12* Symbolics, Inc. 1986.